See Click Predict Fix

Aileen Chen
achen20@stanford.edu

David Ye
davidye@stanford.edu

Introduction

311 is a method for people to file complaints for local issues such as graffiti and potholes for the local government to fix. Citizens may view, vote for, and comment on issues that they are interested in or find important. Municipal governments are interested in predicting the number of views, votes and comments on a given issue, so that they can better respond to and prioritize important problems that people are facing. In this project, we used machine learning techniques to extract features and predict the values of these three counts. We then evaluated our model by comparing our results to other teams' results on the Kaggle data prediction platform.

Data Set

The datasets that we worked with are provided by Kaggle (www.kaggle.com/c/see-click-predict-fix). Each example represents an issue posted to the website and contains the following fields:

1. **Longitude**: the longitude where the issue occured, e.g. "41.314499"
2. **Latitude**: the latitude where the issue occured, e.g. "-72.891544"
3. **Summary**: a short text summary of the issue, e.g. "Very dangerous and deep hole in sidewalk"
4. **Description**: a longer text explanation of the issue, e.g. "This massive hole in the sidewalk continues to trip people and causes many falls and near falls on a daily basis. It continues to get bigger and only gets more dangerous with each passing day"
5. **Source**: a categorical variable indicating where the issue was created, e.g. "iphone"
6. **Time Created**: the time and date when the issue originated, e.g. "1/9/2012 7:14:49 PM"
7. **Tag Type**: a categorical variable (assigned automatically) of the type of issue, e.g. "pothole"

The training examples also contains these target values:

1. **Number of Votes**: the number of user-generated votes
2. **Number of Comments**: the number of user-generated comments
3. **Number of Views**: the number of views

The training set has 223,129 examples that were created between 2012-01-01 and 2013-04-30. The test set has 149,576 examples that were created between 2013-05-01 and 2013-09-17. The distribution of the number of views, votes, and comments in the training set are heavily skewed towards the lower counts, with 49% of the training set having 0 views, 72% having 1 vote (each issue has at least 1 vote), and 96% having 0 comments. There is a long tail in the number of views, with the maximum being 4,381. Similarly, the maximum number of votes is 327 and the maximum number of comments is 86.

Evaluation Metric

The evaluation metric defined by Kaggle was the Root Mean Squared Logarithmic Error:

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(log(p_i + 1) - log(a_i + 1))^2}$$

- n is three times the number of test examples, since there are three values to predict for each issue.
- p_i is our predicted value
- a_i is the actual value

We modified the given dataset by applying the function $f(x) = log(x + 1)$ to the target values. This allowed us to use the more standard RMSE evaluation metric when training our models. To convert back into the original target values, we applied the inverse function $g(x) = e^x - 1$ before submitting our predictions to Kaggle.

Basic Features

We extracted a few basic features from the dataset that were used with little to no processing. The distinction between categorical and continuous features that we describe below only apply when transforming these into binary features.

1. **Source**: Categorical feature. There were 10 distinct sources in the training set.

2. **Tag Type**: Categorical feature. There were 37 tag types in the training set with over 20 samples.

3. **Longitude and Latitude**: Continuous feature.

4. **Creation time**: Categorical feature. We used the day and hour of the week.

5. **Summary**: Categorical feature. There were 87 summaries in the training set that were found in over 100 samples. The summaries were normalized by lower casing and removing punctuation and spaces. These were typically short summaries describing common issues, such as "Tree debris" and "Graffiti on private property".

6. **Description**: Categorical feature. There were 53 descriptions in the training set that were found in ove 100 samples. The descriptions were normalized by lower casing and removing punctuation and spaces. Many of these were descriptions created from automated processes, such as "This issue was reported to the City of Oakland Public Works Agency via phone (501-615-5566), email (pwacallcenter@oaklandnet.com), or web (www.oaklandpw.com)."

7. **Description Length**: Continuous feature. The number of characters in the description.

Advanced Features

We also constructed more advanced features that required additional processing.

1. K-means Location Cluster: Categorical feature. We applied k-means clustering to the longitude and latitude coordinates of the issues to cluster issues together by physical location. With a k = 7, we identified 7 distinct cities / neighborhoods where the majority of the issues came from. The k parameter was determined by running the algorithm with different values for k until we found clusters that appeared physically distinct from each other.

2. Naive Bayes Text Classification Probability: Continuous feature. To capture information from the remaining, unique summary and descriptions, we built a multi-class Naive Bayes text classifier with a multinomial event model and Laplace smoothing. We split the value for views, votes, and comments into buckets. These buckets were used as the classification classes. The words in the summary and description fields were combined to use as features for the classifier. Ultimately, this outputs three integer features for the regression learner. Each integer represents the Naive Bayes bucket prediction for the number of views, votes, and comments.

   When extracting the text features from the summary and description fields, we first converted the text to lowercase and removed punctuation and stopwords. Another consideration was how to create the buckets for the classification classes. A small number of buckets, around three to five, worked best because the predicted posterior probabilities for each individual class weren't too small to be meaningful. We also tried creating equal sized buckets, but this performed slightly worse than creating buckets at equally spaced percentiles in the training counts distributions, since they were heavily skewed towards the lower numbers for votes and comments.

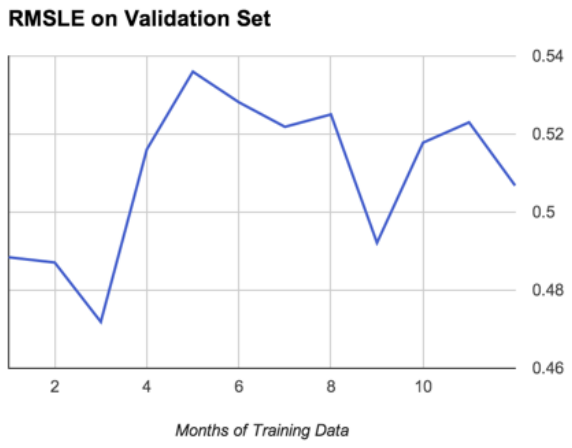
Feature Importance

Using decision trees, we were able to get estimates of feature importances for a set of features. This allowed us to iterate quickly by trying new features and removing uninformative features. These were the feature importances for vote predictions on the final set of features we ended up using:

Feature	Importance
Source	0.2209
Tag Type	0.0067
Longitude	0.0153
Latitude	0.0079
Creation day of week	0.0507
Creation hour of day	0.0209
Summary	0.0102
Description	0.1041
Description Length	0.0089
K-Means Location Cluster	0.0797
Naïve Bayes Text Classification	0.4746

Dataset Pruning

Since our task is to predict future values based on past data, we hypothesized that most recent training examples would be more relevant and that we should prune or discount older data. For example, the number of views, votes and comments may increase steadily over time as the number of 311 users in a community increases. To test this hypothesis and to find the optimal number of months of training data to use, we used the most recent 4 months of training data as our validation set, and trained on the previous 1 to 12 months of data.



Using the most recent 3 months of training data achieved the lowest RMSLE on the validation set. It also improved our RMSLE on the test set dramatically by ~0.07.

Regression Models

Linear Regression

The first model we applied to our data was linear regression. For the categorical features, we transformed each categorical value into its own binary feature. For the continuous features, we transformed each feature by bucketing them into ranges where each range contained roughly the same number of examples. The bucket ids were then used as binary features. This model achieved an RMSLE of 0.37266 on the training set and 0.3187 on the test set.

Support Vector Regression

We also tried support vector regression with an RBF kernel on the same binarized feature set that we used in linear regression. This model achieved an RMSLE of 0.3408 on the training set and 0.3281 on the test set.

Gradient Boosted Decision Trees

Finally we then tried applying gradient boosted regression trees. Since decision trees can naturally segment the feature values for us, we did not need to binarize the features as we did in the previous two models. Using 100 trees with a depth up to 5 for each tree, we were able to obtain a training set RMSLE of 0.3525 and a test set RMSLE of 0.3102.

For all the models the test error was lower than the training error. We hypothesize that this is because the test examples have a different distribution than the training examples, and contain a larger proportion of examples that are easier to predict.

Results Summary

The gradient boosted regression tree model performed the best out of the three models. The official Kaggle competition ended over Thanksgiving break but we were still able to submit results after the competition. Here is how we would have done relative to other teams modeling the same data. For reference, the winner of the competition achieved a test RMSLE of 0.2883.

	Train RMSLE	Test RMSLE	Kaggle Ranking
Gradient Boosted Regression Trees	0.3525	0.3102	54 / 533
SVM	0.3408	0.3281	92 / 533
Linear Regression	0.3727	0.3187	114 / 533